**Amendments to the Specification**

Please replace the paragraph that begins on Page 26, line 16 and carries over to Page 27, line 8 with the following marked-up replacement paragraph:

-- The template concept will now be discussed in further detail, to illustrate how the template-driven approach of preferred embodiments preferably operates. A template is intended to guide the generation process in language-specific ways, as stated earlier. For example, while the template in Figs. 7A through 7F is adapted for generating code in the Java programming language, another template might be developed which creates code for the discoveryURL schema element of Fig. 2 according to nuances of some other programming language. Therefore, the templates used by preferred embodiments are constructed as a general image of the code to be generated. Tags are used within the template code to drive the operation of the generator. In the examples, the template tag syntax has the form "%xxx%", where "xxx" is replaced by some keyword that indicates some particular behavior to be performed by the generator. For example, the "%attribute%" tag, which has been mentioned with reference to Figs. 4A and 4B, signals to the generator that the name of an attribute should be substituted into the generated code in place of this tag. --

Please replace the paragraph that begins on Page 29, line 15 and carries over to Page 30, line 1 with the following marked-up replacement paragraph:

-- Note that the sample class library in Figs. 4A and 4B does not contain a counterpart to the template statements at elements 702 and [[704]] 706, and contains only part of the statements at element 708. It may therefore be assumed that the template which was used for creating that

class library differed from the portion of template 700 that is shown in Fig. 7A. The more complex sample class library in Figs. 8A through 8I, on the other hand, does include these elements (see Fig. 8A), and thus it may be assumed that the template used in generating the more complex sample class library included these elements shown in Fig. 7A. --

Please replace the paragraph on Page 32, lines 1 - 7 with the following marked-up replacement paragraph:

-- The setter and getter methods shown in Fig. 8E and 8G, respectively, highlight the previously-discussed problem whereby manual generation of class library code tends to be error-prone because of its redundant nature. Although many such methods may need to be generated for a particular schema, they differ only slightly. Thus, the generation task will tend to be more accurate if performed programmatically, as when using the present invention. (As stated above with reference to Fig. 4B, the setters [[are]] and getters are generated in response to template code such as that illustrated by elements 780, 782, 784, and 786 of Fig. 7E.) --

Please replace the paragraph that begins on Page 32, line 16 and carries over to Page 33, line 8 with the following marked-up replacement paragraph:

-- The serialization method which begins at 840 of Fig. 8H and continues through the end of Fig. 8I illustrates a more complex serialization approach than that illustrated in Figs. 4A and 4B, and contains code to handle various types of supported element relationships as required for the sample schema fragment in Fig. 6. This serialization method is another part of the output class library where the generated code shown in the example extends beyond what is represented

by the template code. For example, the commentary at 840 of Fig. 8H matches the template commentary in Fig. 7F, and the generated "saveToXML" method signature in Fig. 8I matches the template code at 788; in addition, the three setter methods shown generally at element 845 of Fig. 8I match the pattern for attribute ~~getter~~ setter methods which is specified at element 792 of Fig. 7F. However, the template does not include extensions to support the ~~"saveToXml"~~ "saveToXML" method generation for child elements which corresponding to the group defined in Fig. 6, such as the methods shown at 850 and 855 in Fig. 8I. (Again, it will be obvious from the examples how such extensions to the template can be created.) --

Please replace the paragraph that begins on Page 33, line 17 and carries over to Page 34, line 6 with the following marked-up replacement paragraph:

-- As indicated by Block 905, the input schema which represents the message formats of interest is parsed and is preferably stored in memory. Each parsed element is preferably categorized (Block 910) during the parsing process. This categorization comprises determining the element type, and associating that type information with the element name (in a table or other type of storage structure, preferably). Example categories include: a simple text element which does not allow children; an element allowing a single instance of a child element; and an element which contains a set of child elements (and any sequence restrictions on those child elements is preferably remembered as well). Relationships among elements are remembered, such as which element is a child of another element. Cardinality information for child (i.e. nested) elements is preferably remembered, which may be one of: required; optional; single; or multiple. --

Please replace the paragraph on Page 43, lines 5 - 9 with the following marked-up replacement paragraph:

-- While the preferred embodiments of the present invention have been described, additional variations and modifications in those embodiments may occur to those skilled in the art once they learn of the basic inventive concepts. Therefore, it is intended that the appended claims shall be construed to include both the preferred ~~embodiment~~ embodiments and all such variations and modifications as fall within the spirit and scope of the invention. --